RobotDrlSim: A real time robot simulation platform for reinforcement learning and human interactive demonstration learning

Te Sun, Liang Gong, Xvdong Li, Shenghan Xie, Zhaorun Chen, Qizi Hu, David Filliat

RobotDrlSim: A real time robot simulation platform for reinforcement learning and human interactive demonstration learning

Te Sun, Liang Gong*, Xvdong Li, Shenghan Xie, Zhaorun Chen, Qizi Hu and David Filliat

*Corresponding author email: gongliang_m@sjtu.edu.cn

Abstract. Deep reinforcement learning (DRL) techniques give robotics research an AI boost in many applications. In order to simultaneously accommodate the complex robotic behaviour simulation and DRL algorithm verification, a new simulation platform, namely the RobotDrlSim, is proposed.

First, we design a standardized API interfacing mechanism for coordinating diverse environments on RobotDrlSim platform, where PyBullet simulator is equipped with an API to form a physical engine for robotics simulation. Second, benchmark DRL models are included in the baseline library for evaluation. Third, real-time human-robot interactions can be captured and imported to drive the RobotDrlSim tasks, which provide big data-stream for reinforcement learning. Experimentations show that cutting-edge DRL algorithms developed in python can be seamlessly deployed to the robots, and human interactions can be availed in training the robots. RobotDrlSim is valid for efficiently developing DRL algorithms for artificial intelligence models of robots, and it is especially suitable for the robot educational purposes.

Keywords: Simulation platform, robot control, deep reinforcement learning, human interactive demonstration learning.

1. Introduction

With rapid progress, the intelligent robot draws increasing attention from both academic and industrial fields. Advanced learning models boost robot to achieve complex problems like [1]. Generally, these kinds of self-taught intelligent architecture implement deep learning models for environment perception and knowledge accumulation. Deep reinforcement learning that combines convolutional neural network and bellman dynamic planning not only outperforms human experts in many domains [2][3][4] but also enables robot to be trained in an end-to-end style. However, applying the direct reinforcement learning algorithms to solve complex task would be inefficient, especially within robotic configuration where interactions are lengthy and costly. A typical solution is to use a simulated virtual environment to reconstruct the robot's dynamics and external changes, which substitutes assiduous experiments in real life.

Simulated environments provide models with fast, repeatable and flexible interaction experiences. Growing computation capacity equips simulation with multi-threading, parallel computation and GPU acceleration. Those novel technologies improve the performance of both dynamics simulation and reinforcement learning. On the one hand, high-quality dynamics simulation is used to rapidly prototype robots design and simulate virtual sensors, providing a virtual robotics testing platform. On the other hand, reinforcement learning models will have a more stable and robust manner with large and reliable training data source.

The narrowing gap between simulation and the real-life allows to train a simulated robot and then to deploy the trained reinforcement learning model in real robotic control platform. Yet, the lack of a standardized and plug-and-play robotic simulation solution requires researchers to redesign the platform to fit the current reinforcement learning algorithm interface.
Within the scope of our work, we extend Bullet physic engine [5][6] to build our standardized robotic simulation environment platform, which integrates APIs for reinforcement learning algorithms, remote control and expert demonstration. We proposed two specially designed robotic environments: InMooV[7] and Jaka. Both have their counterpart robots in real-life.

2. Background

Simulation is essential these days for robotic research. The simulated engine provides fast verification for robot design. Meanwhile, it bridges the gap between the reinforcement learning algorithms and the real-world robotic problems. In the work of Plasencia et al[8], they proposed two teaching platforms for robotics simulation. The first of which combines OpenAI Gym with V-REP physical engine, providing a large choice for reinforcement learning-ready API. The second tool they propose uses Deeplearning4J, an open-source deep learning toolbox implemented with JAVA, which offers RL-Glue plug-in. Stephen James[9] uses a 3D simulated robot simulation with V-REP built-in to train a v-rep robot arm to grasp a cubic. To resolve the interface with python, they implement an internet communication protocol to assemble simulated robot and DRL algorithms package. But the time lagging involved with server-client communication prevents online training. Moreover, the survey conducted by Diego Ferigo et al[10] shows that socket-based protocols are discouraged since the existence of multi-components in the simulation makes firstly random initialization partially uncontrollable. Secondly, it may obstruct reproductivity especially when the algorithms are very sensitive to numerical variation. Therefore, client-server solution is not suitable for resolving simulated robots with reinforcement learning.
M. Kirtas et al[11] redesign Webots to fit Gym interface, enlarging the compatibility for both robotic and gaming problems. This open-source toolbox equipped with possibility for more feature extensions offers DRL verification platform for both researchers and students. Yet, the main experiments within their work do not have counterparts in the real-life. And the framework does not contain DRL algorithm customized setups, which leaves some tedious works to track the training and replay the trained model.
Meanwhile, many researchers have completed various tasks in the simulation system by applying the method of reinforcement learning to the simulation. In the work of Jan Matas et al[12], they have used a combination of several deep RL algorithms to solve the problem of manipulating deformable cloth and completed three tasks in the Bullet simulation environment, including folding a towel up to a mark, folding a piece of cloth diagonally and draping a face towel over a hanger. They also transferred the learned policy to the real robot without any real-world data. Chelsea Finn et al[13]. develops a new reinforcement learning algorithm ‘guided cost learning’ based on inverse optimal control. They used the MuJoCo physics simulator to test their algorithm by running experiments on three tasks of varying difficulty, like 2D navigation around obstacles, a 3-link arm reaching towards a goal location in 2D in the presence of physical obstacles. And then they take this result to compare the ‘guided cost learning’ with the prior sample-based algorithm to determine which one is better. Finally, in the research conducted by Deirdre Quillen et al.[14], they build a robotic arm with 7 degrees of freedom in the Bullet simulator with the task of grasping objects from a bin. Then different RL algorithms are applied to the simulation environment. After evaluation, every algorithm will be given with a benchmark to decide which one is the most suitable for grasping situation. However, even with numerous works to apply DRL algorithm in simulated framework, short of readily employed simulation platform poses difficulty for massive benchmarking upcoming state-of-the-art algorithms.
Our RobotDrlSim platform that supports pure Python API abstraction allows direct interaction between the robotic kinematic simulation and the DRL model without socket-based data transmission. The robots in real life also provide possibility to test the simulation accuracy. The stacked 10 DRL algorithms will also facilitate the researcher and students for further development.

3. Methodology

Our platform mainly consists of several task-oriented simulated environments. We have standardized its interfaces to associate with the supported DRL algorithms and the operators themselves. To cooperate PyBullet simulator with DRL algorithms implementation, we migrate Gym API as the wrapper to skip some tedious environment configuration at the very start.
Among the provided environments, our platform contains 2 robotic environments: InMoov and Jaka. Therefore 2 interfaces are built to fit Gym and RL states information with ground truth and raw pixels.

3.1. Build Up the Environment

The simulated environments adopt gym module to interfere with RL algorithm. Within the framework of simulation platform, we integrate several featured functions, some of those are highlighted along with built-in parameters in the Table 1.

Table 1. Parameters and member functions of basis environments wrapper class.

Parameters	**Effect**
urdf_root	*Path to pyBullet urdf files*
Renders	*Whether to display the GUI or not*
is_discrete	*Whether to use discrete or continuous actions*
multi_view	*Whether to return images from multiple camera*
max_distance	*Max distance between the effector and the button*
Functions	**Effect**
getState	*Get the state with a given observation*
Reset	*Reset simulation in PyBullet and initial positions*
get_observation	*Get observation according to the state format*
Render	*Set the cameras and obtain the images*

As for the on-the-shelf SDK configuration of our RobotDrlSim platform, we have already imported the urdf files which describe the InMoov robot with more than 50 controllable joints, and the Jaka robot which has 6. Both of these two robots are representative for their humanoid and cooperative features. For instance, we utilize the *loadURDF()* function compiled in PyBullet toolkit to load InMoov model, materializing the simulated subject as well as some other auxiliary objects or scenes such as *plane.urdf* and *table.urdf* etc.To instantiate our environments as Gym objects, the final step is to register the environment in a registry lists by grouping concerned modules in the script *registry.py*.
On the other side, we constructed reinforcement learning algorithms interface with similar idea. We generate algorithms interface from stable-baselines[15] and include several customized adaptations. Since then, the RobotDrlSim platform can conveniently combine DRL algorithms with registered environments, which render a quick verification on the subjects. Take "ppo2" and Jaka for instance:

Algorithm 1 JakaButtonGymEnv(GymEnv)

```
import PPO2 from stable_baselines
import StableBaselinesRLObject from rl_baselines.base_classes
initializing algorithm PPO2 in the class StableBaselinesRLObject
adding arguments: --num-cpu
```

Algorithm 1: environment-algorithm interaction process for Jaka environment

Finally, the interfaces of client-environments and algorithms in our platform are built. With this platform, robotic simulation with standardized RL algorithms can be easily deployed.

3.2. Training the Model

Everything being prepared, we can train the model whose entrance locates in *rl_baselines.train.py*. Similarly, we need to input some configurational parameters for ArgumentParser to parse. The table below demonstrates some significant inputs. Please refer to *train.py* for the complete list.

Table 2. Key parameters for RL training setup.

Parameters	**Effect**
algo	*RL algorithm to be trained*
env	*environment registered to interact with*
log_dir	*Directory to save agent logs and RL model*
srl_model	*SRL model to use to reduce the dimensions*

During the training process, the corresponding logs will be saved to the directory input as *log_dir* refers to. Later we can observe the results or retrain the model directly via the logs. Naturally, we can utilize the visdom toolkit to supervise the training process in real time.

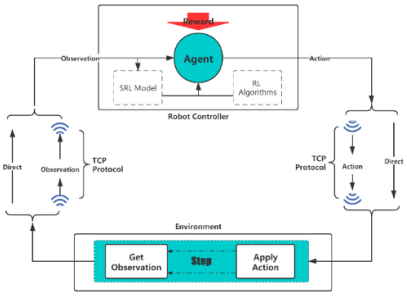

Figure 1. The interaction process of simulator platform and RL algorithm.

3.3. Remote Control

In order to integrate our platform with demonstration learning and take full advantage use of computation resource, we extend the environments to support remote control. Within remote control setup, we separate control and simulation into 2 computers. Through TCP protocol, connection exchanges simulated data and robotic control command. We re-encapsulate the data along with a tokento assure the integrity and reliability of remote data transmission.

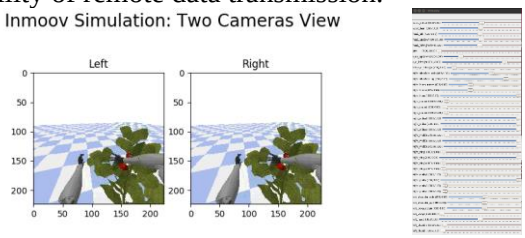

Figure 2. First-perspective camera perspective and joint control from local PC.

Take example for a single time interaction between the local and the server. We firstly initiate socket protocol on both ends: server and client. After establishing connection, GUI will be activated on the local side and corresponding environments will be created on the server side. At one step interaction, local platform broadcasts control command, which will be received by activated RobotDrlSim simulator. The first advantage of this implementation aims to maximize the computational power from server side and to free local computer from complex python configuration. Secondly, most popular real robot control systems like ROS, are designed with C++. The data encapsulation and Internet-styled data transfer protocol offer flexibility of programming languages, which paves the path for direct migration of trained DRL model to real-life configuration. Finally, the implementation allows expert demonstration to support data collection and Learn from Demonstration (LfD) algorithm development.

4. Experimental Demonstration

Our simulation platform contains a collection of ready-to-use environments, including Jaka and InMoov robot. In this part, we investigate the performance of our platform and its compatibility with state-of-the-art DRL algorithms.

4.1. Jaka Robot's Point to Point Control

Jaka robot's point to point task tries to guide the end of Jaka robot to reach the position pre-set by us. We set a button at a fixed position. The goal for the robot is to reach the button. The action space of robot is discretized, containing six possible movements for each time step: moving forward, backward, up, down, left and right. We use pybullet's inverse kinematics algorithms to compute the needed joints-level control movements for robot at each step. In *ground_truth* model, the observation contains the relative position between the end of robot and the button. In *raw_pixels* model, the observation is a photo taken by a global camera we set in the simulation environment. We define *d(robot, button)* as the

distance between robot and button, and for every step taken, the agent is rewarded with *-d(robot,button)*. The learning curves using different algorithms are depicted in Figure 3.

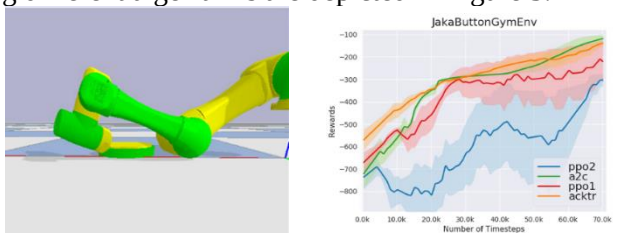

Figure 3. Robot in simulation (left) and learning curves (right) for point-to-point task for Jaka robot

4.2. Jaka robot contacts with the button and avoids obstacles

On the basis of point-to-point training above, an obstacle (a column) is added between the robot and the target button. In this scope, the robot should learn to touch the button while avoiding the obstacle.

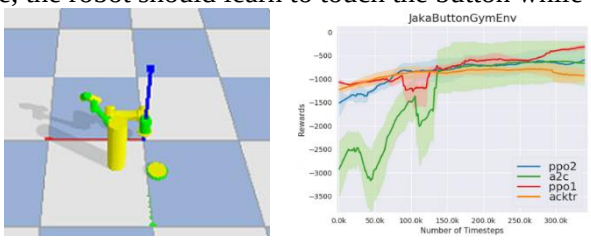

Figure 4. The environment set(left) and the learning curve(right) of obstacles avoiding and reaching

The action space is the same as last mission. The observation contains the relative position between the end of robot and the button and the relative position between the end of robot and the obstacle. For every step taken, the agent is rewarded with *-d(robot,button)*. If the robot gets bumped by the obstacle during training, the agent would get a reward -200. The robot succeeds in making a circular move to approach the button and staying away from the obstacle. The learning curves using different algorithms are depicted in Figure 4.

4.3. InMoov robot's point to point control

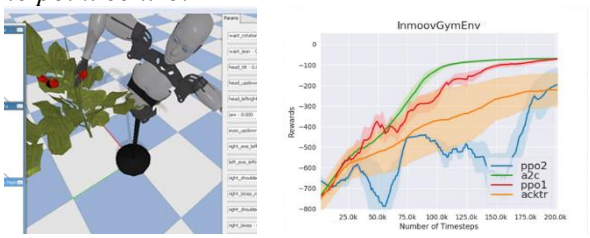

Figure 5. Simulated tomato tree and robot(left); Learning curves of object grasping task for InMoov(right).

InMoov robot's point to point control is meant to let the right hand of InMoov robot reach the position of one tomato on a tomato tree we 'plant' in the simulation environment.

The action space is discrete, containing six possible actions for each time step, the same as that in Jaka point-to-point training. The observation contains the relative position between the right hand of the robot and the tomato. For every step taken, the agent is rewarded with *-d(right hand,tomato)*. The learning curves using different algorithms are depicted in Figure 5.

4.4. Simulation Platform fps Verification

State_model	Jaka	InMoov
Ground_truth	*506*	*76*
Raw_pixels	*22*	*8*

Table 3. The simulation fps for Jaka and InMoov environment.

Experiments on fps defines the number of frames per second the simulator can interact with the environment. The fps of our simulation with different robots and input is shown in the Table 3.

5. Conclusion

RobotDrlSim plays an important role in training robot with deep reinforcement learning methods.
The RobotDrlSim platform breaks the barriers of multiple simulation environment coordination and supports visualization of robot training, which accelerates the deployment of robot control algorithms with complex tasks. Human-robot interactions could also be utilized to guide the robot training, which paves an efficient way for improving the performance of DRL algorithms.
In the next step, we will concentrate on GPU acceleration to achieve real-time simulation with higher image resolution. The future work could also be focused on the deployment of DRL model down to real robot to solve tasks. Moreover, task-specified algorithm and demonstration algorithm could be developed in the future to accelerate the training process.

Acknowledgement

This work was supported by National Natural Science Foundation of China (Grant No.51775333) and the Science Foundation of Shanghai Municipal Commission of Science and Technology (Grant No.18391901000).

References

[1] S. Levine and A. Krizhevsky, "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection," 2012.

[2] O. Vinyals *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019, doi: 10.1038/s41586-019-1724-z.

[3] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016, doi: 10.1038/nature16961.

[4] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," *CoRR*, vol. abs/1312.5, 2013, [Online]. Available: http://arxiv.org/abs/1312.5602.

[5] E. Coumans and others, "Bullet physics library," *Open source bulletphysics. org*, vol. 15, no. 49, p. 5, 2013.

[6] E. Coumans and Y. Bai, "PyBullet, a Python module for physics simulation for games, robotics and machine learning." .

[7] G.Langevin, "InMoov." 2014.

[8] A. Plasencia, Y. Shichkina, I. Suárez, and Z. Ruiz, "Open source robotic simulators platforms for teaching deep reinforcement learning algorithms," *Procedia Comput. Sci.*, vol. 150, pp. 162–170, 2019.

[9] S. James and E. Johns, "3D Simulated Robot Manipulation Using Deep Reinforcement Learning," 2016.

[10] D. Ferigo, S. Traversaro, G. Metta, and D. Pucci, "Gym-Ignition: Reproducible Robotic Simulations for Reinforcement Learning," *Proc. 2020 IEEE/SICE Int. Symp. Syst. Integr. SII 2020*, pp. 885–890, 2020, doi: 10.1109/SII46433.2020.9025951.

[11] M. Kirtas, K. Tsampazis, N. Passalis, and A. Tefas, "Deepbots: A Webots-Based Deep Reinforcement Learning Framework for Robotics," in *Artificial Intelligence Applications and Innovations*, 2020, pp. 64–75.

[12] J. Matas, "Learning end-to-end robotic manipulation of deformable objects," 2018, [Online]. Available:https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1718-ug-projects/Jan-Matas-Learning-end-to-end-robotic-manipulation-of-deformable-objects.pdf.

[13] C. Finn, P. Abbeel, P. Eecs, and B. Edu, "Guided Cost Learning : Deep Inverse Optimal Control via Policy Optimization," vol. 48, 2016.

[14] D. Quillen, E. Jang, O. Nachum, C. Finn, J. Ibarz, and S. Levine, "Deep Reinforcement Learning for Vision-Based Robotic Grasping : A Simulated Comparative Evaluation of Off-Policy Methods," pp. 6284–6291, 2018.

[15] A. Hill *et al.*, "Stable Baselines," *GitHub repository*. GitHub, 2018.